Project Implementation
COS 301 Buzz Project
Group: Testing Phase: Resources
Version 1.0

Carla de Beer 95151835
Prenolan Govender 13102380
Shaun Meintjes 13310896
Collins Mphahlele 12211070
Dumisani Msiza 12225887
Joseph Murray 12030733
Sifiso Shabangu 12081622
Joseph Potgieter 12003672
Johan van Rooyen 11205131

https://github.com/Carla-de-Beer/Testing-Resources

1University of Pretoria

24 April 2015

Contents

1 Introduction

The purpose of this task was to test the functionality provided by the Resources teams for the Buzz Project.

According to the master specification document, version 0.1, released 13 March 2015, the buzzResources module is used to upload and manage resources like media (e.g. images, video) and documents (e.g. PDF documents, Open Document Format documents). These resources are to be either embedded or linked to in posts.

This team took each of the pre and post-conditions of the required use case, considering the work of both Resources A and Resources B, and tested them for compliance with the functional requirements.

The use case for this team is called **uploadResource**. In terms of the requirements for this use case, and as defined by the master specification document, users should be able to upload resources such as media files or documents. Any uploaded resource should be accessible by other users who should also be able to specify links to that resource.

The functional requirements preconditions for this use case included the need to

- detect the mime type
- check that size constraints are met
- check that the resource type is supported

The functional requirements postconditions for this use case included the need to

- check that the resource persisted
- check that the URL for the resource was created

2 Functional Testing

2.1 Resources A

2.1.1 Pre-condition violations

getResourcesBySpaceId The pre-condition will be that the SpaceID that is sent in as a parameter does exist in the database prior to this execution. There are no violations of this pre-condition.

getResourcesAll The pre-condition will be that there exist resources in the database prior to this execution. There are no violations of this pre-condition.

getResourcesRelated The pre-condition will be that the appraisal type (RelatedID) that is sent in as a parameter does exist in the database prior to this execution. There are no violations of this pre-condition.

2.1.2 Post-condition violations

getResourcesBySpaceId The post-condition will be that the resources that exist in a certain space will be returned. This post-condition fails, because the function queries the database and returns all of the resources in the database, which is incorrect. No exceptions are thrown if the Buzz Space does not exist.

getResourcesAll The post-condition will be that all of the resources in the database will be returned. This post-condition holds, and all of the resources are returned.

getResourcesRelated The post-condition will be that the resources that exist in the database of the specified appraisal type will be returned. This post-condition fails, because the function queries the database and returns all of the resources in the database, which is incorrect.

2.1.3 Data structure requirements

getResourcesBySpaceId No data structure requirements mentioned in the service contract, nor does getResourceBySpaceId utilise any data structures.

getResourcesAll No data structure requirements mentioned in the service contract, nor does getResourceAll utilise any data structures..

getResourcesRelated No data structure requirements mentioned in the service contract, nor does getResourcesRelated utilise any data structures.

2.2 Resources B

2.2.1 Pre-condition violations

removeResource The pre-condition of removeResource is assumed to refer to the identification number of a resource in the database which should exist prior to this execution. There are no violations of this pre-condition due to error checks being performed to verify whether or not the entry exists.

2.2.2 Post-condition violations

removeResource The post-condition of removeResource would be the removal of an entry. Due to a lack of unit testing functionality, the post-condition fails as there is no visible indication of removeResource working.

2.2.3 Data structure requirements

removeResource No data structure requirements mentioned in the service contract, nor does removeResource utilise any data structures.

3 Non-functional testing / assessment

3.1 Resources A

The use case was tested against the following list of architectural requirements:

3.1.1 Usability

3.1.2 Performance

3.1.3 Scalability

3.1.4 Testability

3.1.5 Security

3.1.6 Reliability

3.1.7 Reusability

3.1.8 Pluggability

3.2 Resources B

The use case was tested against the following list of architectural requirements:

3.2.1 Usability

removeResource removeResource demonstrates usability by being simple and easy to use. The function only needs be called along with the ID of the resource to be removed which is intuitive.

3.2.2 Performance

removeResource removeResource does not suffer from any performance issues in and of itself, however should the database maintain a sizeable stature, removeResource will be seen to suffer in performance. This is to be expected.

3.2.3 Scalability

removeResource removeResource offers no solutions to accomodate an increase or decrease in size of the database.

3.2.4 Testability

removeResource removeResource has a complete lack of testability due to it not conforming to proper unit testing standards (making use of callbacks, depending on a live database etc.).

3.2.5 Security

removeResource The only security concern would be an incorrect ID being used in order to remove a different resource than required, potentially putting the system at risk.

3.2.6 Reliability

removeResource According to the failure of the post-condition, removeResource gives a poor indication with regards to reliability.

3.2.7 Reusability

removeResource removeResource is not very reusable due to the dependence on a single database. Using this function throughout this system would not be possible/practical.

3.2.8 Pluggability

removeResource In the same vein as reusability, removeResource cannot be integrated into other systems due to a lack of robustness and versatility.

4 Critical Evaluation and Recommendations

4.1 Resources A

4.2 Resources B